Received & accepted second version 9/30/80

Hardware Product Reviews

[Nov 80 - accepted & re-edited for clarity re S-100 bus]

The Casheab Music Synthesizer

by

Jon Bondy

Box 148, Ardmore, Pa, 19003

While at the ~~West~~ Coast Computer Faire in March 1980 ~~In February of this year~~, I discovered the Casheab music synthesizer for the first time. Previous to this, the best music synthesizing equipment ~~which~~ one could purchase for use on the S-100 bus was the Solid State Music SB1 board, a board with distinct limitations (see below). Also at that time, the only music synthesis boards for the Apple were made by ALF, and they only produced square waves. By comparison, the Casheab was extremely versatile and reasonable priced, so I purchased one of their first units, receiving it in June.

To give you an idea of how far things have come in about three years, the SSM SB1 costs about $150 (Kit) and will synthesize one voice with 32 8-bit samples per cycle of the waveform and 15 steps of amplitude control. The Casheab costs about $1000, but provides 32 voices with sixteen waveforms with 1024 12-bit samples per waveform and 255 levels of amplitude control. Although more expensive initially, the Casheab is far less expensive for someone who is serious about creating multi-voice music ~~pieces~~. In addition, the Casheab has FM capabilities, allowing it to do vibrato and more complex FM synthesis, as discussed below.

The standard Casheab synthesizer consists of two S-100 boards linked together by a ribbon cable, one a controller, the other one the synthesizer itself. The former contains the processor interface (S-100), timing generators, an accumulator, and the digital to analog section. The latter contains the frequency generation hardware, waveform memories, and amplitude control hardware. The synthesizer board contains a 16 MHz bit-serial signal processor which scans through the waveform memories at a rate determined by the frequency generation hardware, to produce amplitude samples at a fixed rate through a time-multiplexing scheme. Because of the ribbon cable, the synthesizer cannot be debugged completely without having two extender boards; you can debug each board individually for many problems, however, by removing the jumper. Due to the high frequencies used, the boards are NOT available as Kits, only as assembled and tested units.

The synthesizer has so many control parameters that it is memory mapped in order to avoid tying up most of the I/O ports. It uses 256 bytes of memory for control, usually allocated at 0F800H, although I use 0FF00H. Because all of the memory locations used for synthesizer control are write-only, you can run this board at the same address as a working memory board and the two will not interfere with each other; when the synthesizer is not needed, that area will look like regular memory, and when the memory is not needed, the synthesizer will be available. The synthesizer does asssert the wait lines to allow for internal synchronization, which could cause the regular memory to appear to be slower with the synthesizer in the computer. Inadvertently writing to the memory when the synthesizer is running will have some fairly discordant effects, however.

Each of the 32 channels has a two byte Frequency Control Word (FCW) which controls the rate at which the waveform for the particular channel is scanned. Frequencies can be specified in multiples of approximately 0.3 Hertz from 0 Hertz to about 19 KHz, which provides reasonably precise control for musical purposes. Each channel also has what Casheab calls a 'weight', but which I call an Amplitude Control Word (ACW). These allow each channel to have amplitudes from zero (off) to 255. Each channel also has one byte for selecting which of the sixteen waveform tables it is to use, and a byte to indicate whether it is to FM modulate the channel two above it or not. An FM channel uses its output to increase or decrease the rate at which the channel two above it is scanned, thus increasing or decreasing the pitch of that note. A channel which is used for FM is not heard at the synthesizer output; a non-FM channel is summed with all other non-FM channels and their sum is available at the sound output of the synthesizer for direct connection to a music amplifier. The synthesizer produces a single channel of audio output, combining all 32 channels into one signal.

The waveform tables are loaded by loading a special byte in the memory map with the number of the waveform table to be loaded, and then loading the table data sequentially into another special byte location in the map. One additional special memory location is used for overall scaling of the synthesizer, since the output with all 32 channels in use is significantly greater than with only one channel.

The board is strewn with wire-wrappable jumpers, to allow the user to re-confisure it for either 4 or 16 waveforms, either 1024 or 2048 samples per waveform, and either 10, 16, or 32 channels (yielding sampling rates of

50, 34, or 17 KHZ [for frequency responses of 25, 17, or 8.5 KHZ] respectively). Also, either normal or inverted phase one or phase two S-100 bus clocks can be used to trigger the board, allowing use with all 'standard' processor boards.

The first thing which impressed me about the synthesizer was the care which went into it, in terms of both the quality of the documentation and the boards themselves. The manual is over 70 pages long, and discusses how to install board jumpers to modify the options, how to use the synthesizer hardware, the software which is provided with the boards, theory of operation of the synthesizer (and some other theory too!), references, maintenance procedures, parts lists, and a listing of a sample test program. Schematics and parts layouts are supplied separately. The boards are somewhat densely populated, but the layout is clean, and there are no last minute changes to the PC layout strewn about as is so common with initial production units.

The software is CP/M compatible and is mostly written in Microsoft BASIC, consisting of three main parts, the waveform generator, the score generator, and the Play program. Source code is provided for all software.

The waveform generator uses a Fast Fourrier Transform (FFT -- see "Fast Fourier Transforms on Your Home Computer", W. D. Stanley, S. J. Peterson, Byte, December 1978) to transform user-specified harmonic intensities into a waveform suitable for loading into the synthesizer. Attack and decay envelopes can also be specified (64 values in the range 0-255 each for attack and decay), allowing a particular waveform to be customized into a complete timbre. During my preparation for this article, Casheab suggested

that I try to generate waveforms by simply adding the weighted harmonic waveforms. This turned out to take 7 seconds per harmonic (running under UCSD Pascal -- more on that later), so that ten harmonics took about a minute, as opposed to about 5 minutes with the FFT. The FFT program running under interpreted BASIC takes about 15 minutes to compute a waveform; under compiled BASIC it takes about 5 minutes. Casheab may be supplying such a program with their synthesizer by the time this article is in print. Both harmonics and timbres may be saved on disk.

The score generator accepts score notation as character strings in BASIC DATA statements, and produces a HEX file as output for the Play program. The notes are represented as SANXOTMS, where 'S' represents a possible slur; 'A' the amplitude of the note (0 [off] through 9); 'N' what note (A, B, C, D, E, F, G) is to be played; 'X' whether the note is sharp, flat, or natural; 'O' the octave number (0 through 6); 'T' the duration (time) of the note; and 'M' whether the note is dotted or not. Thus, a 'typical' note might be given as '3F#4Q.-', meaning that with amplitude 3, play an F# in the fourth octave as a dotted quarter note with a post-slur. The number of voices to be scored is specified, as is which channel is to be used by each voice and which voices are FM modulators. The 'notes' for each voice are then listed sequentially, with an 'X' to terminate each voice and an 'E' to terminate the piece. Typographical errors are flagged by the program as errors.

The Play program is the only program written in 8080 assembly language, and it ties the timbres and the scores together. It allows a score to be read into memory and timbres associated with each channel. Channel assignments can be modified, as can FM modulation flags, and attack/decay envelopes can

be scaled. The piece may be started and stopped, and when stopped, the amplitude of the piece and its tempo may be varied. This software works just fine for up to about 5 voices, but for more than that, it is recommended that a real-time clock be available to the Play program in order to produce timing which is truly even. I didn't have a real-time clock and didn't want to purchase one, so I rigged up a 555 timer chip as a variable frequency square wave oscillator controlled by a potentiometer to provide synchronization to the software via an input port. Casheab supplies two versions of their software, one for use with systems with the 8253 real-time clock and one for systems without. Since source is supplied, you could modify the code for the 8253 to work with your own real-time clock.

The procedure for playing a piece is somewhat involved. You first create a series of BASIC DATA statements, using a text editor, to represent the music you want to play. You then run the score program to create a score file. If you need new timbres for the piece, you run the waveform program to generate them. Finally, you run the Play program to hear your music. If an error is made in the score, you must start again at the editor, then the score program, and then the Play program. Despite some inconveniences, however, the software which is delivered with the synthesizer is sufficient to allow one to encode and play any piece of up to 32 simultaneous voices.

Debugging musical pieces in this fashion is very interesting, since the scores are quite like programs, and you must listen to your 'program' to discover the mistakes which you have made. A quarter note which was written as a eighth note will result in one voice 'sliding' earlier by a eighth of a beat for the remainder of the song, usually causing some

discord, and not revealing its exact location in an obvious fashion.

The synthesizer comes complete with the above software and some musical pieces and timbres ready to play. A Bach Fugue and Prelude are included, as is Pictures at an Exhibition and the theme from Star Wars. Casheab also has coded a Bach Two-Part Invention, but it was not on my initial distribution disk. Timbres supplied included trumpet and clarinet, but it is relatively easy to construct new timbres from information in the literature (either Computer Music Journal, or text books on acoustics).

The synthesizer did not work at all when I first plugged it in, but a call to Casheab indicated that my processor (an Ithaca Audio Z-80 board) was one of those which required a modification to the clock phase and sense jumpers. After I removed the jumper from JP15 to JP17 and added a jumper from JP16 to JP17, it worked immediately and correctly. In fact, one surprising thing about this product is that it does EXACTLY what its documentation says it will; not much more, but certainly nothing less. I am used to a certain amount of 'puffing' in my sales brochures, but Casheab delivered exactly what they said, no excuses about "we're still working on it" or some such.

Use of the FM feature probably needs some explanation. Since an FM channel modifies the rate at which the channel two above it is scanned out, an FM channel running at low frequencies can be used to create vibrato in the modulated channel. If one places a sine waveform in channel 0 running at a low rate, say 1 Hz, then the sound coming out of channel 2 will warble slightly as its frequency changes. In order to facilitate this, I modified the syntax of the score program to allow frequencies in the range of 0.3 Hz

to about 12 Hz to be specified directly instead of by using the score program note notation.

A more interesting use of the FM facility is to do 'real' FM music synthesis with it, as described in "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation" by John M. Chowning (Computer Music Journal, Vol 1, No 2). This technique uses a frequency which is a fraction of the carrier frequency as a modulating signal; that is, if you want to hear a 1000 Hz tone, you concurrently modulate it with a 500 or 250 Hz tone. This has the effect of 'spreading' the spectrum of the carrier (1000 Hz) tone so that it has many rich harmonics, even if the two waveforms being considered are simply sine waves! This means that one makes decisions about the characteristics of one's FM timbres by modifying the ratios of the carrier frequency to the modulating frequency, not by changing the harmonic content of either. Additional ways to modify the FM timbres include use of non-sine waveforms for the carrier (but not for the modulator!) and modifying the degree of FM modulation throughout the duration of the note by modifying the attack/decay envelope of the modulating tone.

Other effects which can be obtained include echo and chorusing. Echo can be obtained simply by repeating the notes for a voice on a second channel at a lower amplitude and with a short delay (rest) inserted before the start of the second channel. This effect can be very pleasing with organ fugues, for instance.

Chorusing is an effect which makes you think that more than one instrument is playing a voice. One problem with a digital synthesizer is its

precision: twenty identical voices played at once sounds just like one loud voice. In order for a chorusing effect to work, the voices must play at slightly different frequencies, and so I modified my score program to provide three equally tempered scales, each off from the next by about 2 Hz. This allows me to have up to three channels playing the same voice but with distinct frequencies. Adding a small amount of vibrato (FM) to each channel at a different vibrato frequency allows a reasonable chorusing effect to be obtained. Unfortunately, three FM'ed channels requires six channels for a single voice, making use of these effects somewhat complex and inefficient.

One final effect which I have not yet tried is to use a large number of channels, say eight, to control the harmonics of a single note individually. Using this technique, one can control the amplitudes of each harmonic of the note throughout the duration of the note, allowing very accurate synthesis of real musical instrument voices. Unfortunately, the Casheab could only support four voices which required control of eight harmonics each.

As stated above, I wrote the waveform program in UCSD Pascal, and in fact re-wrote the entire software system in UCSD Pascal, combining it into a single program in order to be able to customize it more easily; only portions of the Play program had to remain in assembly language. The Casheab software takes advantage of single character keystrokes for command selection, just like UCSD Pascal, but it does not take advantage of the random addressing capability of most CRT systems. My new synthesizer software does, and maintains tables of information about the synthesizer and score on the screen at all times. The cumbersome BASIC DATA statement

formats and line numbers were replaced with free-formats and no line numbers. Also added was a screen oriented note editor which allows one to halt the score in the middle of play and see the notes which were then being played displayed on the screen. Those notes and notes near to them in time can then be modified and the score re-played, short-circuiting the laborious edit cycle described above. A channel inhibit feature was also added in order to facilitate debugging multi-voiced pieces. Casheab owners who are interested in running this software can contact me at the address given at the beginning of the article. This software may be the topic of future articles in S-100 Microsystems if time and space permit.

One thing which modifying the software showed me was that the Casheab software does not BEGIN to take advantage of the flexibility which the Casheab hardware could provide. As more people purchase Casheab systems, software should develop to allow really innovative uses of the systems.

One obvious augmentation of the current Casheab system would be to allow a keyboard to be played 'through' it to simulate a sophisticated organ, or better. Casheab is aware of this, and has a general purpose slave processor card (also S-100), which could be used as a smart keyboard scanning card, implemented in wire-wrap form at the moment. It contains a down-loadable Z-80 system, 16K bytes of RAM, with I/O ports and some breadboarding space for placement of multiplexers and cable connectors. Software to run the synthesizer from the keyboard is working at this time, but no product using either this hardware or software has been announced yet.

The current score syntax does not allow for modifications to the tempo of

the piece while it is being played, nor does it allow for 'blue' notes, that is notes which slide between normal equal tempered note frequencies. Also, the current implementation ties a hardware channel to a voice, a restriction which is really unnecessary. With the current software, it is not possible for a note to decay at the same time that another note for the same voice is attacking; that would require one channel to be playing two notes at once. Software to provide dynamic channel allocation would allow this kind of attack/decay overlap.

In addition, the software is written to 'simulate' organ notes rather than percussive notes such as harpsichord or piano. With an organ, the note starts to attack when you hit the key, rises to a sustain level, and stays at that sustain level until the note is released, at which point it decays. With a piano note, the note attacks and then decays for the note duration, a completely different effect. Changes to the attack/decay software will be necessary to fully support percussive instruments.

There is no reason why the Casheab hardware cannot support any of these new concepts, or even more, but the software is not yet available to support them.

All in all, I would say that the Casheab is a high-quality piece of hardware, well thought out, well designed, and well implemented. The software is complete but somewhat spartan, demonstrating the capabilities of the Casheab hardware, but really serving to provide a starting point from which serious computer musicians can depart. It is a unique and reasonably priced S-100 bus board which all computer musicians with S-100 bus systems should investigate.